

# GWT From Scratch – Day 4

## Layouts And Styling

The aim of today's session is to give you an understanding of layout widgets and methods of styling.

### **You will**

- Look at the VerticalPanel and HorizontalPanel
- Find out how to size them
- Add widgets to them
- Set the alignment
- See how to use ordinary CSS
- Use a TabPanel
- Run through the stages of styling the various parts of it
- The look at the Grid
- The FlexTable
- The DockPanel
- Have a quick run-through most of the others
- Round up the ways of styling widgets
  - Built-in style classes
  - Changing style attributes in code
  - Setting Style classes in code
  - Primary, secondary, and dependent styles

### **Questions?**

My ideal is to have no questions at all because everything is perfectly clear, but if you have any questions, then please contact me. I would like to consider the course 'complete' within the limits I have set for it. In other words, I am here to supplement the course if, in any way, it is not well enough explained to get everyone through it. I intend to use the feedback to improve the course and reduce the questions. So any questions are very welcome.

### **Any Problems**

[rx01-day4@examples.roughian.com](mailto:rx01-day4@examples.roughian.com)

© Ian Bambury 2008

## **Layout And Styling**

### **The VerticalPanel**

We came across this back in the first session. It's a CellPanel which means that it is a table-based widget. As the name might suggest, it gives you cells which are arranged vertically.

In a traditional website you might use it to provide an area for a company logo followed by a horizontal menu, then the main paging area, and then a footer of some kind.

You might also use it for things like vertical menus on the left-hand side of the page, or a boxout of special offers or news items on the right-hand side.

It is, as I have said, a table-based widget and therefore anathema to some people, but tables are inherently evil, nevertheless you will have to make your own mind up whether you use them. The alternative is to use divs in your HTML file because the other options are all CellPanel-based, too.

### **The HorizontalPanel**

The HorizontalPanel is just a VerticalPanel on its side, so I shall deal with both these together.

### **Set The Width And Height**

You can set the width with the `setWidth()` method and a height with the `setHeight()` method, or you can do both at the same time with the `setSize()`.

### **Set The Border**

As with all the CellPanels, the `setBorder()` method (which takes an int argument of the width of the outside border in pixels) is often very useful for finding out where the darn thing has gone. Remember also, that in some browsers, with no widget in the cells, the CellPanel (being a table) will not be visible at all even with a border.

### **Adding Widgets**

You've seen this done before. It's a simple

```
panel.add(new Label("Hello"));
```

or

```
Label myLabel = new Label("Hello");  
panel.add(myLabel);
```

depending on whether you need to refer to the label (or whatever it is) that you are adding to the panel.

## Set The Alignment

You can set the horizontal and vertical alignment in code with the `setHorizontalAlignment()` and the `setVerticalAlignment()` methods. They both take parameters from the `HasAlignment` interface, like this:

```
label.setHorizontalAlignment(HasAlignment.ALIGN_CENTER);
```

In Eclipse, for the `HasAlignment`, just type 'hasa' and then hit `Control+SpaceBar` and you will be prompted with a list of two possibilities. You want the first one, so just press enter. Then press the full-stop (period) key and you will be presented with all the options available.

## CSS

All the GWT widgets come with built-in styles. We'll get on to the intricacies of adding your own styles a bit later but for the moment, will have a quick look at the built-in style classes that Google provide.

If you look at Google's documentation for the `Label()`, you will see that it has a style-class of 'gwt-Label'. All you need to do to use this is to add a stylesheet in the HTML file, and at the style in that file.

```
.gwt-Label
{
    border            : 1px solid black;
}
```

Other widgets have slightly more complicated stylesheets. For example,

The `TabPanel`

So let's have a look at the `TabPanel` now. Firstly, we need to add it, and some tabs.

Here's some code.

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TabPanel;

public class Main implements EntryPoint
{
    String text = "This is the content for tab number ";

    public void onModuleLoad()
    {
```

```

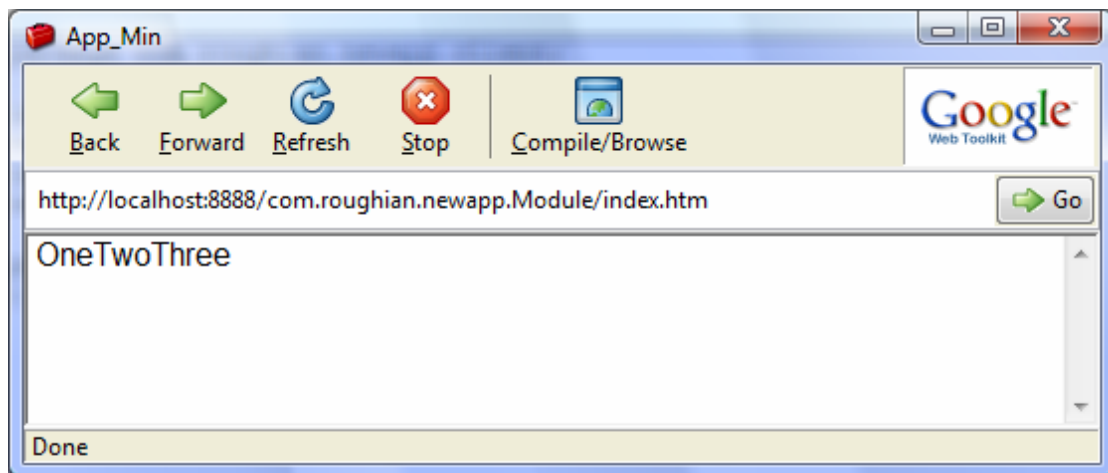
    TabPanel tabpanel = new TabPanel();

    RootPanel.get("slot").add(tabpanel);

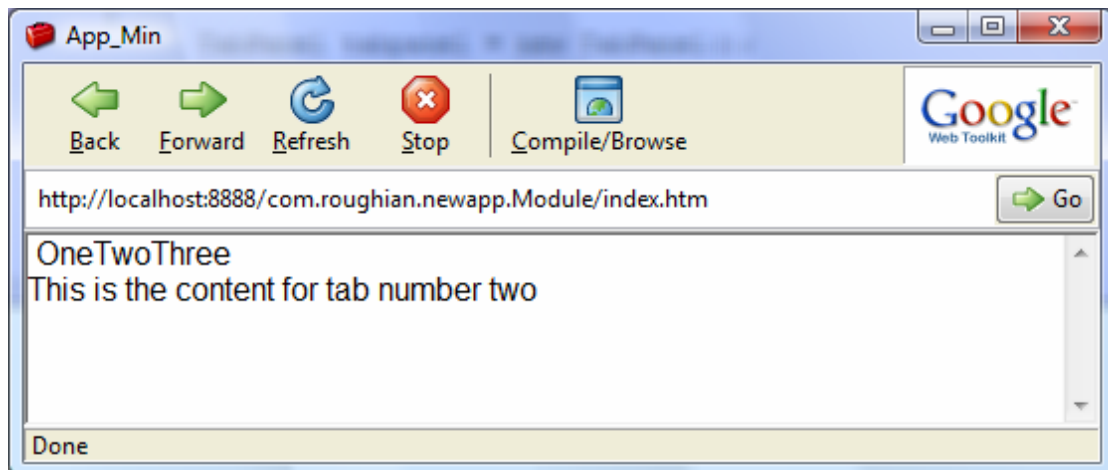
    tabpanel.add(new Label(text + "one"), "One");
    tabpanel.add(new Label(text + "two"), "Two");
    tabpanel.add(new Label(text + "three"), "Three");
}
}

```

The first thing that you'll notice if you run it, is that it looks absolutely terrible. In case you can't be bothered, it looks like this:



Absolutely nothing like a TabPanel, although the text is there. And it does actually work if you click on the word 'Two' then it will display something.



So we need to select a tab before we display it. That is simple enough.

```
tabpanel.selectTab(0);
```

will select the first tab, but we really need to work on the CSS. These are the CSS classes that you get by default from Google.

```
.gwt-TabPanel          /* The whole thing */
```

```

{
}
.gwt-TabPanelBottom          /* The target area for content */
{
}
.gwt-TabBar                  /* The container for the tabs */
{
}
.gwt-TabBar .gwt-TabBarFirst
                             /* A 'dummy' tab at the beginning
                             of the line of tabs to let you
                             add spacing on the left before
                             the first proper tab */
{
}
.gwt-TabBar .gwt-TabBarRest
                             /* All the unused space to the
                             right of the tabs */
{
}
.gwt-TabBar .gwt-TabBarItem
                             /* An unselected tab */
{
}
.gwt-TabBar .gwt-TabBarItem-selected
                             /* Extra CSS and overrides for a
                             selected tab */
{
}

```

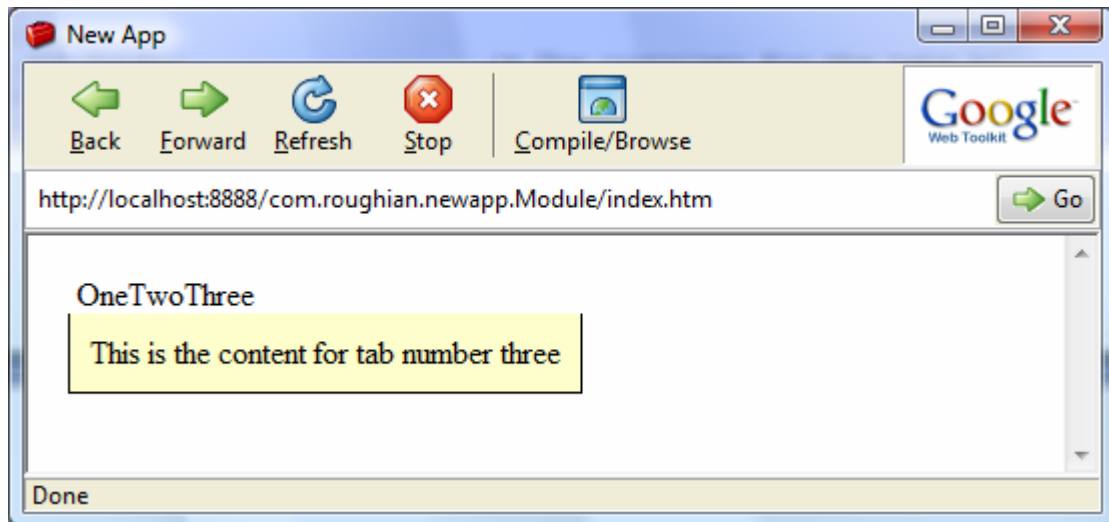
So first I'll push it away from the edge of the BODY element, and give the target area (the 'paging' area) a background colour, some borders, and a bit of padding.

```

.gwt-TabPanel
{
    margin                :    20px;
}
.gwt-TabPanelBottom
{
    background-color       :    #ffc;
    border-left            :    1px solid black;
    border-right           :    1px solid black;
    height                 :    98%;
    border-bottom          :    1px solid black;
    padding                :    10px;
}

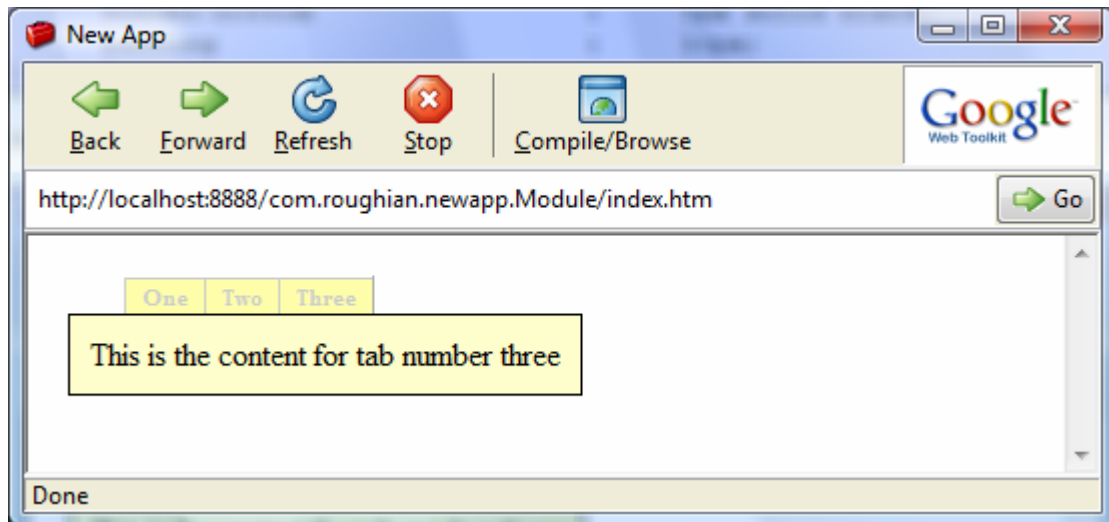
```

And we can see that it is looking a bit better down there.



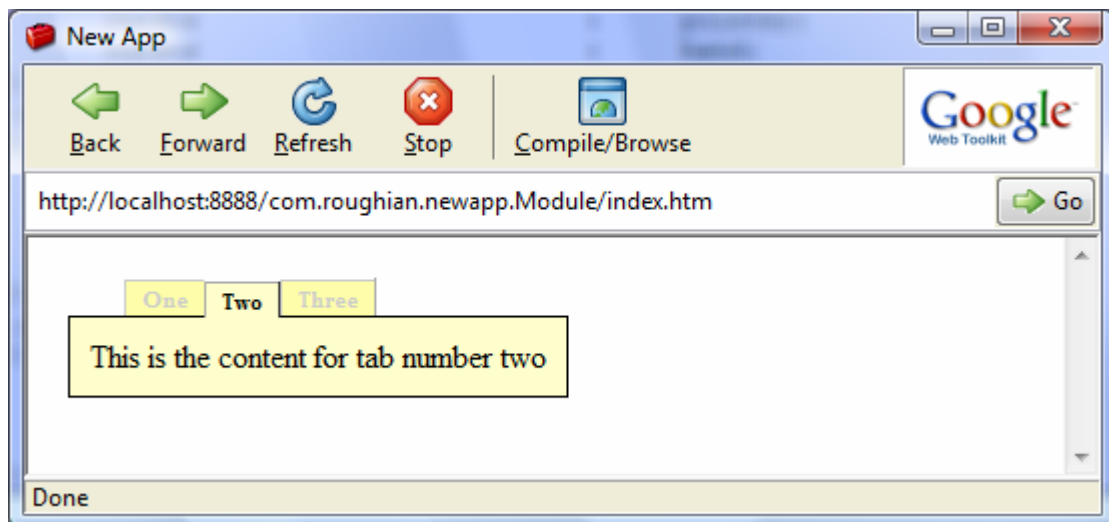
Next we'll have a go at the tabs, the area to their left, and the area to their right.

```
.gwt-TabBar .gwt-TabBarItem
{
    border-top           :    1px solid #ccc;
    border-left          :        1px solid #ccc;
    border-bottom        :    1px solid black;
    background-color     :    #ffa;
    color                :    #ccc;
    font-size            :    70%;
    font-weight          :        bold;
    padding              :    2px 8px 2px 8px;
    cursor               :    pointer;
    cursor               :    hand;
}
.gwt-TabBar .gwt-TabBarFirst
{
    height               :    100%;
    width                :    25px;
    padding-left         :    3px;
    border-bottom        :    1px solid black;
}
.gwt-TabBar .gwt-TabBarRest
{
    border-left          :        1px solid #AAA;
    border-bottom       :    1px solid black;
    padding-right        :    3px;
}
```



And finally, will sort out the highlighting of the selected tab.

```
.gwt-TabBar .gwt-TabBarItem-selected
{
    color                :    black;
    background-color      :    #ffc;
    border-top            :    1px solid #aaa;
    border-left           :    1px solid #aaa;
    border-right          :    1px solid #333;
    border-bottom         :    0px solid black;
    padding               :    2px 8px 2px 8px;
    cursor                :    default;
}
```



Is still very basic, but there are some things you will probably want to take on to your own style. Notice that the bottom border on the selected tab has been set to zero.

This not only joins up a background colour of the tab with the background colour of the paged area, it also drops the text and the top border down which gives the impression that the selected tab is in front of the other two.

That illusion is also helped by the greying of the text in the unselected tabs, and a slight darkening off the unselected tabs background colour.

This is something that many people on the GWT lists complain about (albeit very nicely) by mentioning that it is on a wish list to have a certain amount of basic styling included in the bog-standard version of the widgets.

## The Grid

The Grid is basically a fixed-size table. You declare it to be a certain size, for example

```
Grid widget = new Grid(3, 3);
```

where the '3's are the actual numbers of cells in the x and y directions (and not the highest index you can have).

You can use the `widget.getCellFormatter()...` method to apply various styles to the individual cells.

It has its uses if you are doing some kind of fixed layout like a tic-tac-toe game or have a very rigid layout of some kind. You cannot, however, add cells to the grid on-the-fly, you have to re-declare it and start again.

## The FlexTable

The FlexTable is of course, much more flexible. Have a look at

<http://examples.roughian.com/#Widgets~FlexTable>

where there is a demo. As you can see, you can add rows or cells wherever you please. In the demo you can only add cells to the final row, but in 'real' programming, you can add cells to any row you like. Unfortunately you can only have any ragged edge on the right-hand side, you can't have a ragged edge at the bottom. Which means you can't use it to program games of patience.

Although it looks like it should be a really useful widget, there aren't many times when I have found any use for it except as a fixed width table which you can add to vertically. For this it is ideal, because you don't have to know the number of fields in a record, or the number of records you are going to get back, you can still program for however many turn up.

## The DockPanel

The DockPanel is, in my humble opinion, much overused. It is also not very easy to describe. Imagine you are inside a square room. You can partition that room into two parts by building a new wall inside the room, splitting it into two. So now you have two rooms and you are inside one. You can then build another wall inside the room you are in. And keep repeating this for as long as you like.

Here's a demo



<http://examples.roughian.com/#Panels~DockPanel>

The reason I think this is very overused, is that someone will come up with a design for a website and then try and come up with a way to use the DockPanel to make it work. You end up with cells everywhere.

And you can make it work like that. The problem is (maybe surprisingly) that it is inflexible. It also lures people away from being Object Oriented.

Instead of breaking down a site into the largest pieces possible (for example header, content, footer) and then dividing those into the largest pieces possible, some people will end up with a DockPanel with every single thing they could possibly need, and have to hide some for some pages, and have a general purpose cell here and another one there, just to make it work.

## **Other Panels**

We'll have a quick look at some of the other panels. They all have their uses, some of them obvious like the SplitPanels, and some of them a bit more specialised like the AbsolutePanel.

### **The DisclosurePanel**

This is a panel with a clickable, one-line header which opens or closes a details area.

### **The SplitPanels**

These are panels that are split into two halves. Between the two halves there is an 'splitter' that allows you to resize one cell at the expense of the other in the way that any Explorer program lets you move the divider between the two panes.

There are HorizontalSplitPanels and VerticalSplitPanels.

### **The StackPanel**

A StackPanel is like a stack of DisclosurePanels where only one can be opened at any one time.

See

<http://examples.roughian.com/#Panels~DisclosurePanel>

### **The AbsolutePanel**

AbsolutePanels allow you to position child widgets absolutely. It's just a container for other widgets and you placed these other widgets relative to the top left corner of the AbsolutePanel.

## **The FlowPanel**

The FlowPanel is pretty much the same idea as an AbsolutePanel except that the child widgets are allowed to flow in the normal HTML way.

## **The DeckPanel**

The DeckPanel is what is used for the bottom half of the TabPanel. It can be useful if you need to control the flow of work of a user, for example if you are creating a wizard. You need the user to be able to move onto the next stage, but you don't want tabs at the top of the page so that they can move around at will. In its raw state it's a bit basic and you will need to add controls to let the user move forward or back.

It is very much like cutting a deck of cards, face up. Every time you do it you see a new card, but you can only see one at a time. As soon as you bring one of the widgets in a DeckPanel to the front by selecting it, all the others are, necessarily, hidden.

## **The HTMLPanel**

the HTML panel is a bit of a strange one, but it is quite useful. It is very much like a plain HTML widget where you can add plain HTML, but in the HTML panel, you can add IDs to the elements of the HTML you are adding to the HTML panel (hope you're keeping up with this) and then you can use these IDs to retrieve the elements again.

## **Panel Summary**

That is pretty much it. I covered the last few panels quite quickly because you're probably going into 'Panel Overload'. Don't worry about it, you can always come back and have a quick look through if you need to find the best panel for specific task. You can ask on the GWT user groups, you can have a look through the Roughian site, and if that fails, you can always email me.

## **Styles**

Between version 1.2 and version 1.3, Google changed the way stylesheets worked. This was a bit of a disaster because everyone had got used to the old way, and the old way wasn't available in version 1.3. Not only that, the new way in version 1.3 was really useful for the Google developers who were just developing GWT, but was a pain in the arse if you were just producing applications with the GWT.

For example, in the new version you couldn't actually set the style to something else. You could keep adding styles, and you could take away the styles that you had added. It was a mess. In version 1.4, due to a certain amount of pressure (some of it from me), Google renamed the new commands, and brought back the old commands. There are also some even older, deprecated commands in there, and so the whole thing can be quite confusing.

## GWT Styles

All the widgets have GWT style names in the form gwt-WidgetName. They might also have more of these for the sub-parts of the widget as you saw with the TabPanel (for example 'gwt-TabBar gwt-TabBarItem' which is a TabBarItem within the TabBar widget).

You simply use these in CSS.

## DOM Styles

You saw earlier that you can set the style of an element using the DOM, like this:

```
DOM.setStyleAttribute(label.getElement(),  
                        "border", "1px dotted red");
```

This is useful if you don't want your designers messing around with things that are important to the way your program works. Bear in mind that your graphic designers won't like this at all, and generally it is best to put specific CSS styles in external CSS stylesheets. For one thing it means that you don't have to compile the program if someone says "You know that in dotted red line? I want you to make it blue."

## Setting Style Classes

As well as being able to set styles directly in the code, you can also play around with the style classes. You probably won't be doing this to any great extent until you start creating your own widgets.

There are exceptions, and one of these is when you want to separate a group of widgets visually from another group of the same kind of widgets.

Generally speaking, of course, you will want your site to have an identity. You need one section of it to look similar to another section. You don't want a sudden change of style because that would mean that your visitors would think they'd gone to another site. But that doesn't mean that you might not want a 'theme' to the different sections.

Google, for example, with Gmail, keep the same look and feel for the application, but change the colour scheme if you are looking at emails...

[Dragon NaturallySpeaking thought I said 'if you are looking at females' - chance would be a fine thing at two o'clock in the morning]

... change the colour scheme if you are looking at emails belonging to one of your labels.

It can also be useful to have a style defined for warning messages, error messages, TextBoxes that are in error and so on.

## The Old Style

Let's start with a label.

By default, it has a style class of 'gwt-Label'.

You can use `label.getStyleName()` and it will tell you that.

You can say `label.setStyleName("fred")` and then if you get the style name, it will report 'fred'.

You can say `label.addStyleName("bert")` to get 'fred bert'.

You can say `label.removeStyleName("fred")` to get 'bert'.

That was all there was to it. It was nice and easy and you knew where you were.

## The New Style

Personally, as a beginner, I would leave it at that. Google recommend that you don't use the old-style, but I disagree. Until you need to use a primary, secondary and dependent styles, I wouldn't bother to learn about them. Get the hang of GWT first, and then come back when you find that you can't do everything that you want to do with The Old Style.

It was good enough for everyone including everyone at Google when GWT was at version 1.2, so I don't understand why they are so against it right now. There was so much protest they had to bring back the old-style, so there must be plenty of people out there who think it is adequate for their needs.

But for the sake of completeness, and in case you're interested, here is a quick rundown on the new style.

When you first assign a style to a widget (this is usually done before you get it and so it is already set - like 'gwt-Label'), this style name is designated the Primary Style Name.

If you say `label.addStyleName("bert")` then you will still, of course, get a style of 'gwt-Label bert'. The 'bert' part of the string is known as a Secondary Style Name.

You can also say `label.addSecondaryStyleName("bert")` but this will get you a style of `gwt-Label gwt-Label-bert`

In this case, the 'bert' part is a Secondary Style Name.

This is where it gets interesting. You can now say

```
label.setStylePrimaryName("my-Style");
```

and you will get a style of `my-Style my-Style-bert`

So this is fantastic if you're offering customers all sorts of skins. And it is also terrific if you are creating your own widgets and styling different areas of your website so that they are colour-coded.

Where it causes problems, is if you want to add, say, a yellow background to highlight text, you have to have every darn style defined for every area of your website. In GWT version 1.3, you couldn't add a secondary style name, they had taken that part away. Even now it is back, they don't recommend using it. But I do. You will have to make your own choice. But like I say, my advice is to start with the easy version that many of us demanded come back.

I'm not saying don't use it, it definitely has a place. But while you are coming to terms with all the rest of GWT, why not take the easy route in this one small area?

## **Tomorrow**

### **We Build A Web Site**

As I have said before, elsewhere, it won't be a big website, but we could build it from scratch, it will have three pages (but it will be easy to add more). It will have a history support. And it will have a built-in way of showing pop-up messages while blocking user input, and fading out the background.